CloudSense: Continuous Fine-Grain Cloud Monitoring with Compressive Sensing

CloudSense: Continuous Fine-Grain Cloud Monitoring With Compressive Sensing

H. T. Kung, Chit-Kwan Lin and Dario Vlah
Harvard University
Cambridge, MA 02138
{htk,cklin,dario}@eecs.harvard.edu

Abstract

Continuous fine-grain status monitoring of a cloud data center enables rapid response to anomalies, but handling the resulting torrent of data poses a significant challenge. As a solution, we propose CloudSense, a new switch design that performs in-network compression of status streams via compressive sensing. Using MapReduce straggler detection as an example of cloud monitoring, we give evidence that CloudSense allows earlier detection of stragglers, since finer-grain status can be reported for a given bandwidth budget. Furthermore, CloudSense showcases the advantage of an intrinsic property of compressive sensing decoding that enables detection of the slowest stragglers first. Finally, CloudSense achieves in-network compression via a low-complexity encoding scheme, which is easy and convenient to implement in a switch. We envision that CloudSense switches could form the foundation of a "compressed status information plane" that is useful for monitoring not only the cloud data center itself, but also the user applications that it hosts.

1 Introduction

Status monitoring is an essential component to the smooth operation of today's cloud data centers, as quick responses to anomalies, failures or load have crucial business and performance ramifications. To be economical, clouds must strive for full utilization or high job throughput; thus, it pays to have the freshest server load information to avoid idling. Customer service-level agreements often stipulate system responsiveness requirements that must be met by rapid remediation of failures; this can only be achieved with quick notification via continuous fine-grain status reporting. Recent trends in cloud design and usage patterns further call for fine-grain and low-latency status reporting. For instance, frequent system-level status reports (heartbeats, temperature, network load) are crucial for better decision-making by automated data center management systems [8] and for maintaining geographically distributed, container-based data centers [7], since more subsystem failures can occur as the total system scales up. At the application level, the increasing popularity of NoSQL systems has placed greater emphasis on interactive ad hoc querying, meaning that straggler tasks in the MapReduce jobs underlying NoSQL queries need to be quickly detected and mitigated in order to provide the user with a responsive system.

In short, cloud data centers could greatly benefit from continuous, fine-grain and low-latency global status reports across many dimensions. But storing, transporting and processing the sheer volume of information poses a high data-rate sensing problem. Worse yet, anomaly detection mechanisms must often rely on collecting global status information in order to make global, relative comparisons. For example, straggler detection requires a relative metric since, by definition, straggling tasks are those that run slower than most others. This global information requirement means that data reduction solutions based on local comparisons and filtering are unsuitable. Existing solutions resort to reducing the data volume by either employing aggregation methods [14] to lower the resolution of information or by sampling at a low rate [12]. Unfortunately, neither strategy is amenable to continuous, fine-grain monitoring. This led us to consider an alternative: in-network compression of status messages.

We observe that the required bandwidth for each network link in a status collection tree (Figure 1), including the top link, depends mostly on the "sparsity" of the system, which in our case is the number of anomalies, rather than on a much larger quantity proportional to the total number of nodes. To exploit this, we rely on results in *compressive sensing* [4] (CS), a technique in signal processing that enables simple encoding and exact reconstruction of a *sparse signal* given incomplete samples or *measurements*. In the literature, CS has been considered mainly as a compression technique for signal and imaging problem domains, since natural transforms into sparse domains (e.g., Fourier) are well-known. A major challenge in identifying other areas of applicability has been finding natural sparsity-inducing transforms for other kinds of signals. Here, we show that CS is also useful to discrete applications, such as status monitor-

ing. This is possible because status anomalies are by definition sparse; in other words, the status signal itself directly exhibits sparsity, meaning we can simply use the identity transform.

CS is well-suited for in-network status message compression for two main reasons. First, it provides a simple encoding mechanism for switches at all levels of the network; fan-in at aggregation and core switches can be handled by simple addition operations. Second, it has a useful incremental decoding property—with just a few measurements, the largest anomaly can be recovered first; as more measurements are received, anomalies of smaller magnitude are recovered next. This "largest first" decoding property is perfectly suited for cloud monitoring, as large anomalies are typically revealed earlier to reporting nodes and need to be handled first.

We propose *CloudSense*, a compressive sensing switch design that enables continuous, low-overhead, in-network compression of status reports. Using MapReduce straggler monitoring as an example, we give evidence of the benefits of CloudSense over conventional status reporting methods via analysis and emulation. Ultimately, we envision that CloudSense, together with software APIs, will comprise a *compressed status information plane* for the cloud data center, providing a simple platform for monitoring not only the cloud itself but also the user applications it hosts.

2 Compressive Sensing

A full treatment of compressive sensing is beyond the scope of this paper; Candès and Wakin [5] provide a good review for interested readers. Here, we aim to provide a high-level sketch of the mechanics of CS encoding and decoding and insights into how the technique can be useful in cloud monitoring.

Consider a real-valued, one-dimensional, length-N signal as a vector $\mathbf{x} = \langle x_1 x_2 \dots x_N \rangle$. This signal can be represented as $\mathbf{x} = \mathbf{\Psi s}$ in a predetermined basis $\mathbf{\Psi}$, and is called *K-sparse* if it is a linear combination of only K out of N basis vectors, i.e., only K coefficients in $\mathbf{s}$ are non-zero while the rest are zero, or if it can be approximated by such a linear combination. When $K \ll N$, then the signal $\mathbf{x}$ is said to be *compressible*.

In CS, the signal $\mathbf{x}$ is sampled or *encoded* by a process that produces *measurements* $\mathbf{y} = \mathbf{\Phi x}$. Normally, when $\mathbf{\Phi}$ is a full-rank $N \times N$ matrix, the system of equations is complete and can be solved. However, the case of interest is when $\mathbf{\Phi}$ is $M \times N$, where $M \ll N$, i.e., when the signal is compressed. This is a problem with infinitely many solutions, but compressive sensing theory states that a K-sparse signal $\mathbf{x}$ can be uniquely reconstructed with high probability when $\mathbf{\Phi}$ is a random matrix and when $M \geq cK \log(N/K)$, where c is a small constant. The reconstruction, or *decoding*, is usually performed

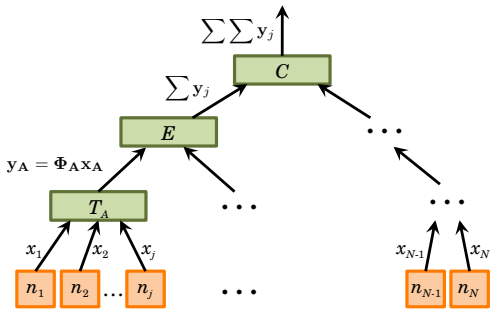

Figure 1: A status information collection tree where nodes n_i are reporting entities under CloudSense monitoring. T_A, E and C are CloudSense switches. This represents a simplified data center network topology.

via a linear programming optimization that solves the ℓ_1-minimization problem:

$$\min_{\mathbf{s} \in \mathbb{R}^N} \|\mathbf{s}\|_{\ell_1} \quad \text{subject to} \quad \mathbf{y} = \mathbf{\Phi x}, \mathbf{x} = \mathbf{\Psi s} \qquad (1)$$

An interesting property of the ℓ_1-minimization is that the quality of the decoding is a function of M. In general, the larger the M, the more accurate the reconstruction. Furthermore, recovery is incremental: with small M, the largest components of $\mathbf{s}$ can be recovered, but as M grows, the remaining components are decoded.

The low complexity of encoding and the flexibility of incorporating any $\mathbf{\Psi}$ in decoding make CS a potential solution to cloud status monitoring. Indeed, CS has been proposed in monitoring data center temperature [11]. In this paper, we extend the idea to new classes of applications related to data centers, including, e.g., CPU load monitoring for VM load spreading, per-flow bandwidth monitoring for heavy-hitter identification or even spam/DDoS detection. As mentioned earlier, we observe that CS is especially useful in scenarios where the notion of a "normal" status is relative and can only be determined by obtaining global information. One such example is in MapReduce straggler monitoring, which we will discuss in Section 4.

3 CloudSense Switch Design

The design of our CloudSense switch prototype dovetails with recent work on programmable switches, such as the SideCar [13] and ServerSwitch [10] projects. In fact, for our prototype implementation, we assume a hardware setup similar to SideCar: a CloudSense switch is comprised of a commodity switch connected to a general-purpose sidecar processor on a specially-designated port.

Figure 1 shows a typical data center topology with CloudSense switches and N rack nodes. Raw status messages (x_i) sent by rack nodes (n_i) are marked with a custom CloudSense IP protocol ID and a status type.

Each status type defines a reporting interval d and is user-specified. On arrival at a top-of-rack (TOR) switch (e.g., T_A), these packets are steered to the sidecar processor, which runs a CloudSense daemon that buffers messages of the same status type into epochs of duration d. Every epoch, the buffered messages in vector $\mathbf{x_A}$ are encoded as $\mathbf{y_A} = \mathbf{\Phi_A x_A}$, where $\mathbf{\Phi_A}$ is an $M \times N_A$ random matrix unique to T_A with N_A being the number of nodes under T_A. (Recall that for status monitoring applications, $\mathbf{\Psi}$ can be the identity matrix. In this case, $\mathbf{s}$ is simply $\mathbf{x}$.) $\mathbf{y_A}$, containing M coded measurements, is forwarded to an end-of-row/aggregation CloudSense switch E, which performs the same kind of packet steering as T_A. Encoding at E, and subsequently at core switch C, is simple summing of $\mathbf{y}_j$. In a more optimized design, a CloudSense switch can perform summing operations with hardware, right at the Ethernet ports.

Note that CloudSense requires neither synchronization amongst switches or rack nodes, nor reliable delivery of measurements. While status may be reported in one epoch but not the next, CloudSense TOR switches always buffer the latest report and thus sends the freshest report available. This fundamental robustness against loss is another advantage of CS. Finally, CloudSense is able to achieve low-latency because CS encoding operations have such low complexity, meaning latency is just a function of the depth of the data center tree.

4 MapReduce Straggler Monitoring

We use MapReduce straggler monitoring as an example scenario for CloudSense. Stragglers are often present in MapReduce jobs and can significantly prolong job completion times [2], reducing both job throughput and responsiveness. As jobs become even more parallelized and shorter (e.g., Hadoop plans to support >200,000 cores [1]) and as users expect ever-faster response, rapid straggler detection has become an increasingly important component of various mitigation methods (e.g., speculative pipelining [9]). A conventional monitoring approach would need to gather $O(N)$ status reports to determine relative task progress and detect stragglers. In contrast, a CS approach would need just $O(K \log(N/K))$ reports, where K is the number of anomalies. We expect $K \ll N$.

4.1 Emulation Methodology and Results

We evaluated CS for MapReduce straggler monitoring using the following discrete-time emulation strategy. First, we generated traces of nominal task progress reports from an emulated MapReduce phase, split across $N = 8000$ nodes with one task per node. Each node's trace was generated by reading a large file (1GB) from its local disk and reporting its progress at each time step (100ms). This captures task progress jitter due to independent, random interrupts from other system processes on each node. Next, we designated $K = 800$ (10%) of the nodes to be stragglers and artificially dilated each of their progress traces by some factor S_i, where K and the S_i's are drawn from distributions published by Ananthanarayanan *et al.* ([2], Figure 2). In practice, $1.5 \leq S_i \leq 10$ and we assume, for simplicity, that each straggler progresses at a constant rate before task completion. Finally, we emulated status messages arriving at a single TOR CloudSense switch by aligning the traces and considering each time step as a signal vector $\mathbf{x}$. At each time step, the emulated CloudSense switch encoded $\mathbf{y} = \mathbf{\Phi x}$, resulting in M measurements. Below, we refer to the decoding of each such time step a separate "CS instance".

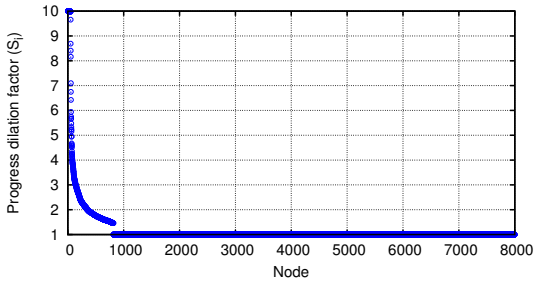

Figure 2: A sorted straggler signal vector $\mathbf{x}$. The signal is the ground truth task duration of all 8,000 nodes in the emulated traces. There is an abrupt drop at 800 nodes because the traces were generated with $K = 800$ stragglers and all other nodes were considered normal. Signals such as this, which are sparse and have rapidly dropping tails, can have their largest components decoded early, with just a few measurements.

Here, it is instructive to note that CS decoding has three failure conditions. First, CS decoding is only successful with high probability. However, CS permits more frequent reporting, meaning a rare decoding failure can be quickly corrected by decoding a subsequent message. Second, CS decoding can fail if the magnitudes of the sparse signal components do not rise significantly above the signal noise (i.e., the measured signal is not sufficiently sparse). This can occur, e.g., when there are insufficient statistics at the beginning of a MapReduce phase to reliably report progress. Third, even if progress is reliably reported, the recovered solution may not identify all stragglers correctly when M is too small. Fortunately, CS may still recover large anomalies in this case.

Figure 2 illustrates a signal vector $\mathbf{x}$, used in our evaluations below, involving 8,000 nodes and sorted according to each node's ground truth task duration. Since severe stragglers with long durations constitute a small percentage of the total population, the signal is sparse. Moreover, the signal tail drops rapidly, meaning the number of measurements required to accurately decode the largest magnitude anomalies is low [3] and *decoding*

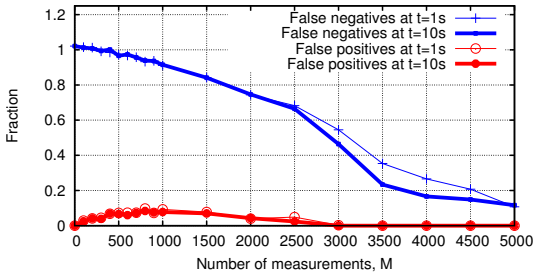

Figure 3: Decoding accuracy vs. number of measurements at two representative points in time. The performance at later times was not significantly different from that at $t = 10$s. The decoding accuracy is computed as the number of false positives and negatives, when compared to the ground truth.

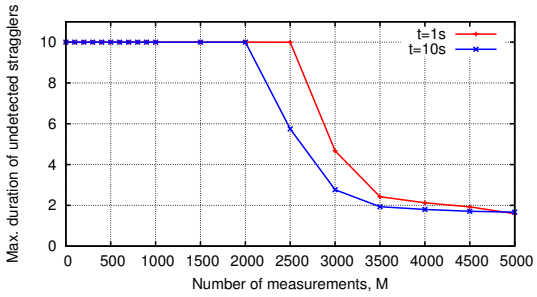

Figure 4: Maximum duration of undetected stragglers vs. number of measurements M. The duration is calculated relative to the median; e.g., a value of 8 indicates a task which runs 8x as long as the median. Two representative time steps are shown; beyond $t = 10$s the error behavior does not change significantly. The flat portions of the curves at small M are due to the task duration distribution we used to generate the ground truth; the maximum duration present there was 10.

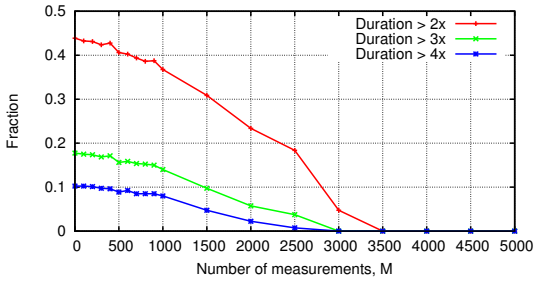

Figure 5: At $t = 10$s, the fraction of undetected stragglers (i.e., false negatives) whose durations are 2, 3, and 4 times greater than the median task duration, respectively.

can occur early, as soon as the first few measurements are received. In Evaluations 1 and 2 below, we are interested in knowing the effect of M on decoding accuracy for such signals, even when M is relatively small. That is, we want to characterize how the tail behavior of the straggler signal governs the accuracy of the decoded solution for a given M in order to give a sense of how useful this early decoding property is.

Evaluation 1: Decoding Accuracy

We first evaluate CS decoding accuracy. In any given CS instance, we compared the set of stragglers identified by CS to the known set of stragglers in the ground truth by computing two quantities: 1) false positives, i.e., the number of nodes incorrectly identified as stragglers by CS, and 2) false negatives, i.e., the number of nodes that were stragglers, but which CS failed to identify.

In Figure 3, we plotted these quantities for a range of values for M, at two representative points in time. There are two noteworthy observations; first, the number of false positives is relatively small regardless of the M value. Thus, when the number of measurements is insufficient for accurate decoding, CS errs conservatively by identifying just a few nodes as stragglers. Second, as M increases, we see that the number of false negatives decreases at a consistent rate, indicating that by tuning M, we can control the accuracy over a wide range.

Evaluation 2: Early Detection of Worst Stragglers

Decoding accuracy alone does not tell the whole story on the effectiveness of CS in straggler monitoring. A more relevant metric is the *maximum duration of undetected stragglers*; even if complete detection is not achieved, CS could be deemed successful if it can identify the most egregious stragglers. Figure 4 shows the maximum durations of the undetected stragglers for the same set of CS instances as in the previous section.

Even though the decoding may not be exact, as M increases, CS tends to detect the more extreme stragglers first; as a result, the maximum duration of undetected stragglers *drops off relatively early*. This property, characteristic of compressive sensing, is useful because it matches well the goal of our application. With fewer measurements, CS identifies just the largest—ostensibly most important—stragglers. With more measurements, CS can identify the remaining stragglers of progressively smaller magnitudes. This is behavior is clearly illustrated in Figure 5, where we have further broken down the undetected stragglers at CS instance $t = 10$s into those that have task durations greater than 2, 3 and 4 times the median task duration. With small M, most of the worst stragglers (4x) are successfully detected leaving the remainder to comprise a relatively small fraction of the undetected straggler population. As M increases, the fraction of worst stragglers (4x) declines to near zero earlier—at $M \approx 2,500$—as opposed to at $M \approx 3,500$ for less severe stragglers (2x).

The largest-straggler-first property suggests that an iterative method might perform even better in straggler removal. In particular, one might use CS with small M at first to identify a few of the slowest stragglers, *remove them from the input*, and then have an easier time identifying the remainder in the subsequent iterations. Under a separate paper, we have developed a theory for this it-

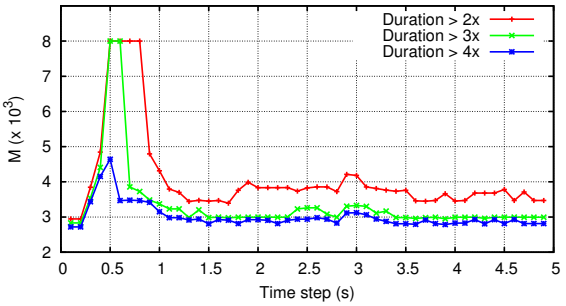

Figure 6: Number of measurements M needed over time to identify stragglers whose durations are larger than 2, 3, or 4 times the median duration, respectively. Only the first 5s of the emulated job is shown. Data points before 0.5s are artifacts related to task start time jitter and should be ignored.

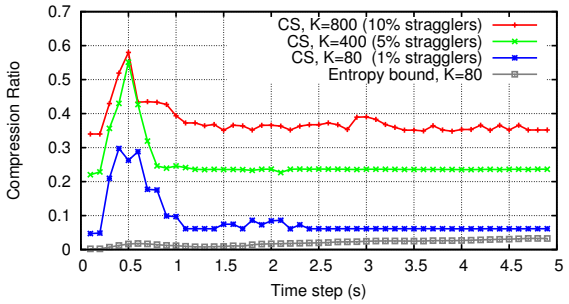

Figure 7: The CloudSense compression ratio improves as the percentage of straggling nodes decreases, i.e., as the sparsity of the straggler signal increases. For comparison, we show the lower bound on compression given by the entropy of the trace where only 1% of nodes are stragglers. In this case, CloudSense approaches the bound.

erative decoding [6].

Evaluation 3: CS Signal Recovery Over Time

We next examine the performance of CS over time by reporting the number of measurements per time step needed to eliminate all stragglers which exceed a specified maximum duration. Figure 6 shows the resulting M values for several maximum durations.

The main feature of the results are the peaks in M at $t \leq 1$s. There are three major factors contributing to the observed shape:

1) Before the onset of the peak, M starts out small, indicating that the progress vector is sparse. This explained by jitter in task start times; in the beginning, a minority of the tasks start early, reporting a non-zero progress, while the majority of tasks have yet to start, and so report zero progress. This constitutes a sparse input to CS, although the detected anomalies are not stragglers at this point.

2) During the peak itself, M is large, indicating a lack of sparsity. This is explained by the fact that most tasks have started by now, reporting a small amount of progress. The stragglers' progress at this point is too small to stand out from the jitter noise; thus, the sparsity in the input vector is weak, requiring a large M to decode at the required level.

3) Finally, after the peak, M decreases again, indicating that inputs are again sparse. We can explain this by the fact that stragglers have grown enough to stand out.

It is evident that the earliest time we can detect most of the stragglers using a low number of measurements is at the end of the peak in Figure 6, when the stragglers "reveal" themselves. Given a higher reporting frequency of CS, on average we can find such points much earlier than with standard RPC reporting as in Apache Hadoop.

Evaluation 4: Compression Ratio

Finally, we consider the compression performance of CS. Since the major advantage of CS lies in its unique ability to detect the largest magnitude anomalies early, we define the CloudSense compression ratio to be the number of messages needed to detect the worst (4x) stragglers relative to the $M = 8,000$ required by Hadoop's RPC method. Figure 7 ($K = 800$, red) shows the compression ratio of CS over the first 5s of our emulated MapReduce job, for the case where stragglers comprise 10% of nodes. Given the same bandwidth budget, CloudSense allows ∼2.7x more status messages to be sent, or a 60% shorter reporting interval.

This compression ratio can be improved significantly when there are fewer stragglers, since the number of messages required by CS to decode is dependent on the sparsity K (i.e., the number of stragglers) of the signal. Figure 7 illustrates the improved compression ratios when stragglers comprise only 5% ($K = 400$, green) and 1% ($K = 80$, blue) of nodes. In these cases, CloudSense can send ∼4.2x and ∼16.3x more status messages, respectively. For comparison, Figure 7 also shows the compression lower bound (grey), as dictated by the entropy of the 1% straggler signal; CloudSense can approach the bound in this case.

Such compression ratios could also be achieved by conventional compression methods such as entropy coding. However, these methods suffer from several significant qualitative drawbacks. First, unlike CS, entropy coding requires correlation amongst messages near the sources in the routing topology to achieve low compression ratios; maximizing this would require jointly optimizing routing and compression. Second, standard algorithms such as Lempel-Ziv are more difficult to implement in switches than CS encoding, even on programmable ones [13, 10]. Third, at tree fan-in points, conventionally compressed messages must either be decompressed and recompressed in order to be combined (which introduces latency) or be simply forwarded (which wastes bandwidth). In contrast, CS encoding can be performed at all levels of the network tree quickly.

Finally, CS offers unique advantages, such as "largest first" partial decoding, that conventional methods do not.

5 Conclusions

We have presented a novel use of compressive sensing: in-network compression of data center status messages. We argue that compressive sensing is a natural fit for this problem domain for two main reasons: (1) its low-complexity compression scheme enables continuous, fine-grain and low-latency cloud status monitoring; and (2) its "largest first" partial decoding property allows for early detection of the most severe anomalies. Additionally, the false positives that arise from such partial recovery do not severely impact the performance of cloud applications, such as MapReduce jobs. In the context of MapReduce straggler monitoring, we have developed a framework to analyze the tradeoff between communication costs (M) and decoding accuracy/straggler improvement. Finally, our proposed CloudSense switch design illustrates that compressive sensing provides a convenient, low-complexity encoding method for in-network compression that is simple to implement.

6 Towards a Compressed Status Information Plane

We envision a general monitoring service, or *compressed status information plane* for the cloud data center, comprised of three components. First, *a centralized registry* would track each status message type (e.g., by allocating unique IDs), inform CloudSense switches of each type's requisite reporting interval and act as a lookup service for type discovery by monitoring applications ("monitors" for short). Second, *CloudSense switches* would encode multiple status streams of different types at the same time, with each stream possibly being tapped by multiple monitors simultaneously. Third, *a software API* would permit monitors to look up status types of interest in the centralized registry and to tap into those streams at any switch in the data center network. Different monitors tapping into the same status stream, but requiring different levels of status fidelity, can be supported naturally by exploiting CS's "largest first" decoding property: those requiring less fidelity could simply collect fewer measurements and decode earlier than those requiring higher fidelity.

Within the rubric outlined above, standard status messages such as CPU or network load could be registered by the data center operator as default status streams. But, as we have illustrated using MapReduce, cloud software infrastructures and even user applications could be instrumented to report statuses (e.g., performance counter values) in a standard way via the compressed status information plane. We believe this general framework could open up new possibilities for performance monitoring of cloud-hosted applications in the future.

Acknowledgments

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-10-2-0180. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. The authors would like to thank the Office of the Secretary of Defense (OSD/ASD(R&E)/RD/IS&CS) for their guidance and support of this research.

References

[1] http://developer.yahoo.com/blogs/hadoop/posts/2011/02/mapreduce-nextgen/.

[2] ANANTHANARAYANAN, G., KANDULA, S., GREENBERG, A., STOICA, I., LU, Y., SAHA, B., AND HARRIS, E. Reining in the Outliers in Map-Reduce Clusters using Mantri. In *OSDI* (2010).

[3] CANDÈS, E. J. The restricted isometry property and its implications for compressed sensing. *Comptes Rendus de l'AcadÂ©mie des sciences, Paris*. Series I, vol. 346, pp. 589, 2008.

[4] CANDÈS, E. J., ROMBERG, J., AND TAO, T. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Trans. Info. Theory 52*, 2 (2006), 489–509.

[5] CANDÈS, E. J., AND WAKIN, M. B. An Introduction To Compressive Sampling. *IEEE Sig. Proc. Mag. 25*, 2 (2008), 21–30.

[6] CHEN, H.-C., AND KUNG, H. T. Separation-based joint decided in compressive sensing. In *ICCCN* (2011).

[7] GREENBERG, A., HAMILTON, J., MALTZ, D. A., AND PATEL, P. The cost of a cloud: research problems in data center networks. *SIGCOMM Comp. Comm. Rev. 39* (2008), 68–73.

[8] ISARD, M. Autopilot: automatic data center management. *SIGOPS Op. Sys. Rev. 41*, 2 (2007), 60–67.

[9] KUNG, H. T., LIN, C.-K., VLAH, D., AND BERLANDA SCORZA, G. Speculative pipelining for compute cloud programming. In *MILCOM* (2010).

[10] LU, G., GUO, C., LI, Y., ZHOU, Z., YUAN, T., WU, H., XIONG, Y., GAO, R., AND ZHANG, Y. Serverswitch: A programmable and high performance platform for data center networks. Tech. Rep. MSR-TR-2011-24, Microsoft Research Asia, 2011.

[11] LUO, C., WU, F., SUN, J., AND CHEN, C. W. Compressive data gathering for large-scale wireless sensor networks. In *MobiCom* (2009).

[12] MASSIE, M. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing 30*, 7 (2004), 817–840.

[13] SHIEH, A., KANDULA, S., AND SIRER, E. G. SideCar: building programmable datacenter networks without programmable switches. In *HotNets* (2010).

[14] VAN RENESSE, R., BIRMAN, K. P., AND VOGELS, W. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comp. Sys. 21*, 2 (2003), 164–206.